A Generalized Digraph Model for Expressing Dependencies

Pascal Fradet, Xiaojie Guo, Jean-François Monin, Sophie Quinton

A Generalized Digraph Model for Expressing Dependencies

Pascal Fradet[1], Xiaojie Guo[1,2], Jean-François Monin[2] and Sophie Quinton[1]
[1]*Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, F-38000 Grenoble France*
[2]*Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, F-38000 Grenoble France*

ABSTRACT

In the context of computer assisted verification of schedulability analyses, very expressive task models are useful to factorize the correctness proofs of as many analyses as possible. The digraph task model seems a good candidate due to its powerful expressivity. Alas, its ability to capture dependencies between arrival and execution times of jobs of different tasks is very limited.

We propose here a task model that generalizes the digraph model and its corresponding analysis for fixed-priority scheduling with limited preemption. A task may generate several types of jobs, each with its own worst-case execution time, priority, non-preemptable segments and maximum jitter. We present the correctness proof of the analysis in a way amenable to its formalization in the Coq proof assistant.

Our objective (still in progress) is to formally certify the analysis for that general model such that the correctness proof of a more specific (standard or novel) analysis boils down to specifying and proving its translation into our model. Furthermore, expressing many different analyses in a common framework paves the way for formal comparisons.

1 INTRODUCTION

The need for computer assisted verification of analysis techniques in the area of real-time systems has been recognized by the research community. The work presented here is part of an effort to contribute to Prosa [1], a library of definitions and proofs for real-time schedulability analyses using the Coq proof assistant [2]. Formal verification requires an important human effort, so making proofs general, generic, and/or reusable is of great importance.

Our goal is to factorize the formal certification of existing Response Time Analyses (RTAs) for fixed-priority policies. There exists a wide variety of task models and analyses for such policies. However, most of these models are incomparable and very few can describe dependencies between arrival and/or execution time of different jobs. The Digraph Real-Time Task (DRT) model [21] seems a good candidate for modeling intra-task dependencies, but its ability to capture dependencies between different tasks is very limited. It cannot, for example, capture Tindell's offset model [25].

RTNS '18, October 10–12, 2018, Chasseneuil-du-Poitou, France

ACM ISBN 978-1-4503-6463-8/18/10…$15.00
https://doi.org/10.1145/3273905.3273918

In this paper, we propose a task model that generalizes the DRT model and its corresponding RTA, with the restriction that we consider discrete time while some results about DRT apply to dense time. A task may generate several types of jobs, each with its own worse-case execution time (WCET), priority, non-preemptable segments and jitter. Our model can capture dependencies between jobs of the same task as well as jobs of different tasks. We focus on fixed-priority scheduling policies and our model can encompass preemptive and nonpreemptive models, as well as limited preemption. Despite being much more general, the RTA for our model is not significantly more complex than the original one. Also, it underlines similarities between existing analyses, in particular the analysis for the DRT model and Tindell's offset model.

For the time being, the proof of the RTA of the general model within the Coq proof assistant is not yet complete. When it is certified, the design and proof of a more specific (standard or novel) RTA will boil down to specifying and proving its translation into our model. Furthermore, expressing many different RTAs in a common framework paves the way for formal comparisons and generalizations (*e.g.,* design of novel RTAs).

To summarize, the main contributions of this paper are:

(1) A general task model which encompasses complex dependencies between jobs and tasks;
(2) A RTA for that model;
(3) A correctness proof of that RTA amenable to its formalization in Coq and applicable to other task models.

The paper is structured as follows. Section 2 introduces the systems we consider and provides basic definitions to describe their runtime behavior. Section 3 presents the syntax and semantics of our task model and Section 4 provides intuition about its expressivity. Section 5 presents the associated RTA and Section 6 sketches its correctness proof. Section 7 puts the contribution of this paper into the perspective of our broader project of a Coq library of schedulability results. We discuss related models in Section 8 and conclude in Section 9.

2 SYSTEM BEHAVIOR

We target concrete systems implemented as a set of tasks executing on a uniprocessor. The execution proceeds according to a *job-level fixed-priority limited preemptive* (JFPLP) scheduling policy. A JFPLP scheduler arbitrates between jobs competing for processor time by choosing the highest priority job, but it can only preempt a running job at some predefined execution point. In other words, each job is decomposed into non-preemptable segments. This model subsumes the Fixed Priority Preemptive (FPP) and Fixed Priority Non-Preemptive (FPNP) policies, and permits mixed policies [3]. A task can generate different types of jobs. Jobs of the same type have similar properties, in particular they have the same priority.

2.1 Definition of system behavior

We assume a set $\mathbb{T}$ of type names (types, for short). Each type entails a number of characteristics that are described in Section 2.

Definition 1 (Job⋆[1]). A job $\jmath$ is specified by:

- its *type* $\mathbf{v}(\jmath) \in \mathbb{T}$;
- its *priority* $\mathbf{p}(\jmath) \in \mathbb{N}$ inherited from its type; A greater number means a higher priority.
- its *arrival time* $\mathbf{a}(\jmath) \in \mathbb{N}$;
- its *jitter* $\mathbf{j}(\jmath) \in \mathbb{N}$ (also called release delay);
- a vector $\vec{\mathbf{c}}(\jmath) = \langle c_1, \ldots, c_s \rangle$, $c_i \in \mathbb{N}^+$, of durations corresponding to the *cost* (*i.e.,* execution time) of each non-preemptable segment.

The *cost* (or required service time) of a job $\jmath$ as above is $\mathbf{c}(\jmath) = \sum_{1 \leq i \leq s} c_i$. The *release time* of $\jmath$ is $\mathbf{r}(\jmath) := \mathbf{a}(\jmath) + \mathbf{j}(\jmath)$.

We do not exclude different jobs from having the same parameters, but we assume that they can be distinguished (*e.g.,* through an identifier). We also assume that the set of jobs is partitioned into *tasks*. The behavior of a system is described using a set of executions defined by a *job arrival sequence* and a *schedule*.

Definition 2 (Job arrival sequence⋆). A *job arrival sequence* is a function ρ mapping any time instant t to a finite (possibly empty) set of jobs $\rho(t)$ such that $\jmath \in \rho(t)$ iff $\mathbf{a}(\jmath) = t$.

Note that once the set of jobs and the function $\mathbf{a}$ are given, ρ is uniquely determined. A job $\jmath$ *completes* when it has received as much service time as it required, which is determined by the schedule. We denote its completion time by $\mathbf{end}(\jmath)$. The *response time* of $\jmath$ is defined as $RT_\jmath := \mathbf{end}(\jmath) - \mathbf{a}(\jmath)$. From its release time and until completion, a job is said to be *pending*.

Definition 3 (Schedule, JFPLP schedule⋆). A *schedule* is a partial function σ which maps any time instant t to the job (if any) that is scheduled (*i.e.,* receives service) at t. A job $\jmath$ can be scheduled only when it is pending.

A *JFPLP schedule* is a schedule such that the job that is scheduled is: either the job that is already executing one of its non-preemptable segments, or a job that has the highest priority h among pending jobs. If there are several pending jobs with priority h, a task is arbitrarily selected among those with such jobs; the chosen job is then the first released job with priority h in this task (FIFO policy).

2.2 Additional definitions and notations

In the following, we will need additional definitions and notations, which we introduce here.

Definition 4 (Job release sequence). Let ρ be a job arrival sequence. The corresponding job release sequence, written $\hat{\rho}$, is defined as $\hat{\rho}(t) := \{\jmath \mid \jmath \in \rho(t') \wedge t = t' + \mathbf{j}(\jmath)\}$. We have $\jmath \in \hat{\rho}(t)$ iff $\mathbf{r}(\jmath) = t$.

We will occasionally write arrival (or release) sequences as lists of sets of jobs. For instance, $[(t_1, \{\jmath_1, \jmath_2\}), (t_2, \{\jmath_3\}), \ldots]$ denotes a sequence ρ such that $\rho(t_1) = \{\jmath_1, \jmath_2\}$, $\rho(t_2) = \{\jmath_3\}$ and $\rho(t) = \emptyset$ for all t absent from the list.

[1]The definitions and lemmas decorated with a star have been formalized in Coq/Prosa, see discussion in Section 7.

The *restriction* of a job arrival sequence ρ to a time interval $[t_1, t_2[$ is denoted $\rho/_{[t_1, t_2[}$. The same applies to job release sequences.

Definition 5 (Workload⋆). Let ρ be a job arrival sequence and V a set of job types. The *workload* $wl_{V,\rho}$ of jobs with type in V in a time interval $[t_1, t_1 + \Delta[$ is the cumulative cost (*i.e.,* required service time) of such jobs released in that interval. Formally,

$$wl_{V,\rho}(t_1, \Delta) := \sum_{\substack{\jmath : \upsilon \in V \\ t_1 \leq \mathbf{r}(\jmath) < t_1 + \Delta}} \mathbf{c}(\jmath) \tag{1}$$

Definition 6 (Service time⋆). Let σ be a schedule and V a set of job types. The *service time* $serv_{V,\sigma}$ received by jobs with type in V in a time interval $[t_1, t_1 + \Delta[$ is

$$serv_{V,\sigma}(t_1, \Delta) := \sum_{\substack{t \in [t_1, t_1 + \Delta[\\ \mathbf{v}(\sigma(t)) \in V}} 1 \tag{2}$$

Our RTA analysis (Section 5) is based on the concept of busy window which we formally introduce now. The following definitions are implicitly parameterized by a job arrival sequence ρ and a schedule σ.

Definition 7 (Level-p quiet time⋆). An instant t is said to be a *level-p quiet time* if all jobs of priority higher than or equal to p released strictly before t have completed at t.

Definition 8 (Level-p busy window⋆). A time interval $[t_1, t_2[$ is said to be a *level-p busy window* if:

(1) t_1 and t_2 are level-p quiet times;
(2) there is no level-p quiet time in $]t_1, t_2[$; and
(3) at least one job with a priority higher than or equal to p is released in $[t_1, t_2[$.

The last condition excludes degenerate cases of busy windows in which no job is scheduled. Since several jobs with the same type may be released in the same busy window, we introduce (again in a way similar to the state of the art) the additional concept of queueing prefix.

Definition 9 (Queueing prefix⋆). The *q-th queueing prefix of jobs of type υ* in a level-p busy window $[t_1, t_2[$ is the time interval $[t_1, t_q]$ where t_q is the instant at which the last non-preemptable segment of the q-th job of type υ receives its first service (*i.e.,* is scheduled for the first time).

Example 1. Figure 1 presents a job arrival sequence ρ (split into ρ_1 and ρ_2) in a system made of two tasks. One task produces jobs of type υ, priority 1, jitter 0 and segments $\langle 2, 2 \rangle$ and two consecutive arrival times of its jobs are separated by at least 9 time units. The other task produces jobs of priority 2. The three first queueing prefixes of jobs of type υ in the level-1 busy window $[0, 27[$ are $Q_{\upsilon,\rho}(1)$, $Q_{\upsilon,\rho}(2)$ and $Q_{\upsilon,\rho}(3)$.

3 GENERALIZED DIGRAPH TASK MODEL

Our task model, called the *generalized digraph* (GD) model, is an extension of the digraph model [21] with job level priorities, possibly null inter-arrival times, jitter and non-preemptable segments.

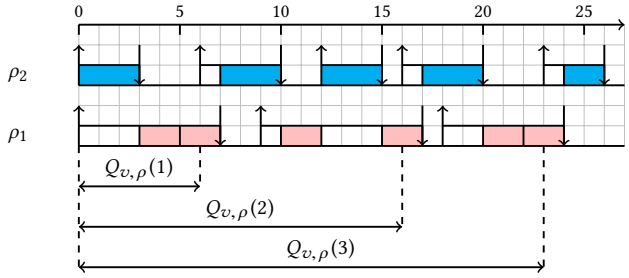

Figure 1: Queueing prefixes of jobs of type v.

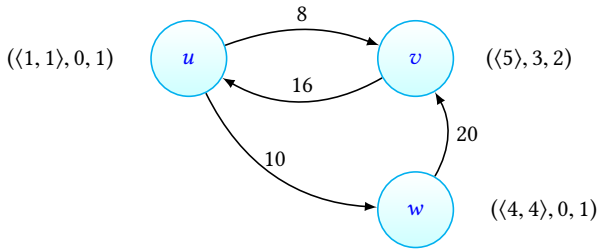

Figure 2: A graph G_e specifying a task with 3 types of jobs

3.1 Syntax

A system consists of a set of n independent tasks $\Sigma := \{G_1, \ldots, G_n\}$, each task being specified by a graph $G_i := (V_i, E_i)$ where:

- V_i is a set of vertices representing different job types;
- E_i is a set of edges such that an edge connecting two vertices v_1 and v_2 of V_i is labeled with a duration $d(v_1, v_2) \in \mathbb{N}$ representing the minimum inter-arrival time between jobs of types v_1 and v_2.

A job type v is characterized by the following parameters[2]:

- $\mathbf{P}(v) \in \mathbb{N}$ defines the priority of jobs of type v;
- $\mathbf{J}(v) \in \mathbb{N}$ specifies the maximum jitter (*i.e.,* delay between arrival and release time) for jobs of type v;
- $\vec{\mathbf{C}}(v) = \langle C_1, \ldots, C_s\rangle$ is a vector specifying the maximum cost of each non-preemptable segment of jobs of type v; $\mathbf{C}(v) = \sum_{i=1}^{s} C_i$ defines the maximum cost of jobs of type v.

The sets of vertices V_i are assumed to be disjoint (*i.e.,* tasks activate jobs of different types). The set of vertices of the complete system is denoted by $V_\Sigma = \bigcup_{i=1}^{n} V_i$. For simplicity and to improve readability, we assume in this paper that the jitter is *constrained*[3]: the jitter of any vertex $v \in V_\Sigma$ is smaller than or equal to the minimum inter-arrival time labeled on any edge going out of v.

In contrast with the standard digraph task model, null inter-arrival times are allowed. We however disallow tasks (*i.e.,* graphs) that contain null cycles (which would permit an infinite nomber of job arrivals at the same instant). This is easily verified statically.

In the following, we note $hep(p)$, $hp(p)$, $ep(p)$, $lp(p)$ the sets of vertices of the system whose priorities are equal or higher, higher, equal and lower than p, respectively.

Example 2. The graph G_e represented in Figure 2 defines a task with three vertices (*i.e.,* three types of jobs). Vertex u is decorated with the triplet $(\langle 1,1\rangle, 0, 1)$ where $\langle 1,1\rangle$ indicates that jobs of this type can be preempted at each instant (segments of length 1) and their maximum cost is $2 = 1 + 1$; their maximum release jitter is 0 and their priority is 1. Similarly, jobs of type v have a maximum cost of 5 and cannot be preempted, their maximum jitter and priority are 3 and 2, respectively. Jobs of type w have a maximum cost of 8 and can be preempted after at most 4 time units of execution.

3.2 Task-level and system paths

As a graph, a task $G_i = (V_i, E_i)$ specifies a set of (possibly infinite) paths, i.e., sequences of vertices of G_i such that $(v_j, v_{j+1}) \in E_i$. Note that a task will typically have cycles, thus specifying an infinite set of paths, some of them infinite. In the following, we sometimes refer to paths of tasks as task-level paths for clarity.

Example 3. In Figure 2, $[u, v, u, w, v]$, $[w, v, u, v, u]$, and the infinite sequence $X = [u, v, X]$ are paths of G_e, but $[u, v, w]$ is not.

Definition 10 (System path)**.** A *system path* of system Σ is a set $\pi := \{\pi_1, \ldots, \pi_n\}$ such that for each i, π_i is a task-level path of task G_i in Σ. The set of system paths of Σ is denoted Π_Σ.

We will use later the following operations on task-level paths.

- The function *len* returns the sum of all minimum inter-arrival times on the edges of a finite path. Formally:

$$len([v_1, v_2, \ldots, v_k]) := \sum_{1 \le j < k} d(v_j, v_{j+1})$$

- The function $pre_\Delta(\pi_i)$ returns the longest prefix π_p of π_i such that $len(\pi_p) < \Delta$. A variant, written $pre^n_v(\pi_i)$, returns the prefix of π_i up to the n-th occurrence of vertex v in π_i.
- We write $len^n_v(\pi_i)$ for $len(pre^n_v(\pi_i))$ which returns the sum of all minimum inter-arrival times between the first vertex and the n-th occurrence of vertex v in π_i.
- The function $cost(seq)$ returns the sum of the maximum cost of all vertices in a vertex sequence seq.
- The filter function $|\pi|_V$ returns the vertex sequence obtained from π where all vertices not belonging to V have been filtered out.

3.3 Semantics

The semantics of a system $\Sigma := \{G_1, \ldots, G_n\}$ is given by the set of arrival sequences that are *consistent* with a system path of Σ. Consistency between an arrival sequence and a system path ensures that jobs in the sequence satisfy the constraints imposed on them by their type and that the order and timing of job arrivals is compatible with the constraints specified on the edges of the graphs G_i.

Definition 11 (Consistency of an arrival sequence *w.r.t.* a path)**.** An arrival sequence ρ is *consistent* with a task-level path $\pi_i = [v_1, v_2, \ldots]$, which is denoted $\rho \sim \pi$, iff there exists a flattening[4] $[(t_1, J_1), (t_2, J_2), \ldots]$ of ρ such that for all k:

- J_k is consistent with v_k, *i.e.,* :
  - $\mathbf{v}(J_k) = v_k$;
  - $\mathbf{p}(J_k) = \mathbf{P}(v)$;
  - $\mathbf{j}(J_k) \le \mathbf{J}(v_k)$; and

[2]The RTA presented in this paper does not rely on *deadlines* and we omit this parameter.
[3]The extension to arbitrary jitter is discussed in Section 4.3.

[4]For example, the arrival sequence $[(t_1, \{J_1, J_2\}), (t_3, \{J_3\})]$ can be flattened into either $[(t_1, J_1), (t_1, J_2), (t_3, J_3)]$ or $[(t_1, J_2), (t_1, J_1), (t_3, J_3)]$.

– $\vec{\mathbf{c}}(J_k) = \langle c_1, \ldots, c_s \rangle \wedge \vec{\mathbf{C}}(v_k) = \langle C_1, \ldots, C_s \rangle \wedge c_i \leq C_i, i = 1 \ldots s.$

- $d(v_k, v_{k+1}) \leq t_{k+1} - t_k$

The definition naturally extends to system-level paths.

We write $\rho \sim \Sigma$ to denote that an arrival sequence ρ is consistent with a system path in Π_Σ.

4 EXPRESSIVITY OF THE GD MODEL

In this section, we show how a variety of existing task models can be expressed using the GD model. We also hint at extended or new models that could be defined as GD and analyzed by our proposed RTA.

4.1 Models without task dependencies

The GD model can easily emulate many kinds of arrival models. Obviously, a simple sporadic task whose jobs are of type v and with a minimum inter-arrival time p is represented by a single vertex v and self-loop labeled with p. A periodic task of period p can be represented by the same GD task. Of course, this GD task represents many more consistent job arrival sequences but the worst case analyzed by the RTA is precisely the periodic sequence.

Arrival curves [24] represent more expressive arrival models. For instance, the minimal distance function $d^-(a)$ returns the smallest time interval that may contain $a+1$ occurrences of a job. This function must be super-additive *i.e.,* $d^-(a) + d^-(b) \leq d^-(a+b)$. In general, an arrival curve may have an infinite description in terms of GD task (*e.g.,* $d^-(a) = a^2$ is super-additive and describes ever growing inter-arrival times). However, such functions[5] are usually given by a collection of values from 1 to some constant k. Then, the analysis uses the minimal super-additive extension (*e.g.,* the smallest function compatible with $d^-(1), \ldots, d^-(k)$). Such super-additive closures can be represented faithfully by a GD task. When d^- is convex (*i.e.,* $d^-(x+1) + d^-(x-1) - 2d^-(x) \geq 0$) then the minimal super-additive extension is periodic and $\forall x = qk + r, d^-(x) = qd^-(k) + d^-(r)$. The corresponding GD task is then made of a cycle of k vertices $v_0, \ldots, v_{k-1}$ where the minimum inter-arrival time decorating each edge is $d(v_i, v_{(i+1) \bmod k}) = d^-((i+1) \bmod k) - d^-(i)$.

Example 4. Consider, for instance, the arrival curve specified by $d^-(1) = 2, d^-(2) = 5, d^-(3) = 10, d^-(4) = 20$ which specifies that 2 (resp. 3, 4, and 5) jobs cannot arrive in less than 2 (resp. 5, 10 and 20) time units. The corresponding GD task is given in Fig.3 (a).

For non convex functions, it has been shown that their minimal super-additive extensions are pseudo-periodic functions [10] which can also be represented by a finite GD task.

4.2 Models with job and task dependencies

As a generalization of digraphs, the GD model can of course express all task models with intra-task dependencies that standard digraphs can. Let us cite, the multiframe model [18] and its generalized version [6], the recurring branching [7] or recurring RT [8] models, the non cyclic RT [4] and digraph model (DRT) [21].

Allowing job-level fixed priorities and null minimal inter-arrival times allows the GD model to model inter-task dependencies

[5] By default $d^-(0) = 0$

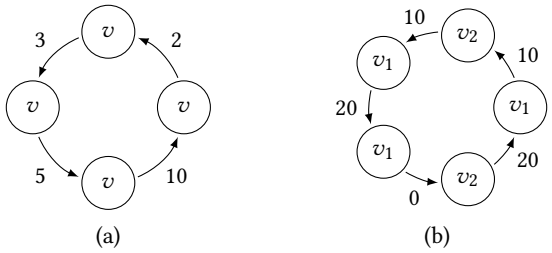

Figure 3: GD representing (a) an arrival curve model and (b) a transaction with offsets *a la* Tindell)

(*e.g.,* fixed timing relation among tasks), *e.g.,* the offset model of Tindell [25] (which cannot be represented in the standard DRT model).

In Tindell's model, a system is made of a set of independent transactions $\{Tr_1, \ldots, Tr_n\}$. Each transaction Tr_i consists of a set of periodic tasks with offsets $Tr_i := \{\ldots, \tau_{i,k}, \ldots\}$ where each task $\tau_{i,k}$ has its own WCET, period, offset, priority, and deadline. A transaction and its set of periodic tasks with offsets can be represented within a single GD task. It is sufficient to compute the hyper-period of the transaction and to build the circular GD task representing the inter-arrival time (which may be null) between arrivals of the different jobs in the hyper-period. The periods and offsets are represented by inter-arrival times. The WCET, priority and deadline of tasks are represented by the corresponding vertices.

Example 5. Consider, for instance, a transaction Tr with two periodic tasks with jobs of type v_1 and v_2, with periods 20 and 30 and with offsets 5 and 15. The hyper-period is 60 and taking the first arrival of v_1 as the time origin, the arrival times are $[(v_1, 0), (v_2, 10), (v_1, 20), (v_1, 40), (v_2, 40), (v_1, 60)]$. The transaction Tr is represented by the GD task in Fig.3 (b). Its worst job arrival sequence *w.r.t.* the RTA is exactly the job arrival sequences of Tr.

Shared resources, which entail inter-tasks dependencies, is a common issue in hard real-time systems. Abdullah et al. [3] addressed this problem using DRT by allowing two kinds of vertices : preemptable (tasks) and non-preemptable (resources). They proposed an extension of the RTA to take into account these two kinds of vertices. Using the GD model, this is directly modeled using segments with always preemptable vertices for tasks and non-preemptable ones for resources.

Rendez-vous mechanisms, another kind of inter-task dependencies, have been expressed using an extension of digraphs (SDRT) [17]. We believe that the encoding of such inter-task synchronization is possible in GD tasks but the exact encoding as well as its complexity remain to be investigated.

4.3 Beyond existing models

All these features, including jitter, can be combined to express and analyse new task models. For instance, non-preemptable segments and jitter have not been considered into intra-task dependencies task models (*e.g.,* MF, GMF, RB, RR, DRT) nor do Tindell's model consider intra-task dependencies (*e.g.,* if a transaction could be a set of MF tasks instead of simple periodic tasks). Since all these

models can be expressed as the GD model, they can be analyzed using the RTA described next.

Note that the following RTA targets the GD model with constrained jitters. Because any GD task with arbitrary jitters can be transformed into a GD task with constrained jitters remaining the same worst case analyzed by the RTA. For instance, a sporadic task of the minimum inter-arrival time 10 and jitter 22 represented by G can be transformed into G' with jitter 0 in Fig. 4.

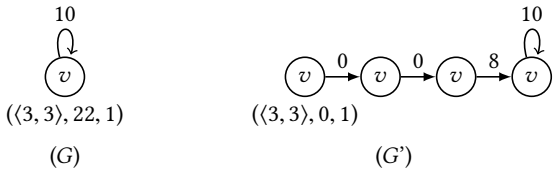

Figure 4: Encoding of arbitrary jitter

5 RESPONSE TIME ANALYSIS OF GD SYSTEMS

The objective of RTA is to bound, as tightly as possible, the *worst-case response time* of each vertex (job type) v, defined as the maximum response time among all jobs of type v occurring in all arrival sequences consistent with each possible system path. Formally,

$$wcrt(v) := \max\{RT_J \mid J : v \wedge \exists \pi \in \Pi_\Sigma, \exists \rho \sim \pi, J \in \rho\}$$

In this section, we present the overall structure of the analysis that we propose for GD systems. We suppose given a vertex v, with priority p, of a task $G_i \in \Sigma$ and focus on upper bounding its worst-case response time.

Our analysis relies on a path-specific analysis of level-p busy windows, hence the following definition (implicitly parameterized as before by a job arrival sequence ρ).

Definition 12 (Busy window path)**.** A level-p busy window $[t_1, t_2[$ is said to be *represented by* the shortest system path π such that $\hat{\rho}/_{[t_1,t_2[} \sim \pi$, and any extension of it.

The shortest path representing a busy window $[t_1, t_2[$ is the path obtained after restricting the job release sequence to $[t_1, t_2[$. In other words, a busy window path is an abstraction of a level-p busy window. Equipped with these notions, the general principle of our RTA analysis can now be presented.

5.1 Overall structure of the RTA of GD systems

The RTA of a vertex v with priority p, of task G_i, consists in analyzing a set of paths such that all possible level-p busy windows are represented by one path in the set. The methodology to bound the worst-case response time of jobs of type v is thus as follows.

Step 1: Derive a set of system paths $\mathbf{\Pi}^v_\Sigma$ such that any possible level-p busy window (for any arrival sequence and schedule consistent with any system path in Π_Σ) is represented by one path in $\mathbf{\Pi}^v_\Sigma$.

Step 2: For each system path $\pi \in \mathbf{\Pi}^v_\Sigma$:

a) Compute an upper bound $BW^+_{p,\pi}$ on the length of any level-p busy window represented by π;

b) Derive from π_i (the task-level path of G_i in π) and $BW^+_{p,\pi}$ an upper bound $q^+_{v,\pi_i,BW^+_{p,\pi}}$ on the number of jobs of type v released in any level-p busy window represented by π;

c) Compute, for each $q \leq q^+_{v,\pi_i,BW^+_{p,\pi}}$, an upper bound $Q^+_{v,\pi}(q)$ on the length of the q-th queueing prefix of jobs of type v in any level-p busy window represented by π;

d) For each $q \leq q^+_{v,\pi_i,BW^+_{p,\pi}}$, compute a lower bound $\theta^-_{v,\pi_i}(q)$ on the (possibly negative) time difference between the q-th arrival of a job of type v in in any level-p busy window represented by π and the start of that busy window;

e) Based on the above, compute an upper bound $RT^+_\pi(v)$ on the worst-case response time of any job of type v in any level-p busy window represented by π.

Step 3: Finally, compute an upper bound $RT^+_\Sigma(v)$ on $wcrt(v)$.

In the following subsection, we provide the formulas used for the computations associated with each step of the methodology that we just presented, along with some intuition of where they come from. The proof of correctness of the computed values is presented in Section 6.

5.2 Step-by-step RTA of GD systems

Our RTA relies on an upper bound on the workload of a set of jobs in any level-p busy window by the *workload for the path* in $\mathbf{\Pi}^v_\Sigma$ that represents it, where the workload of a given set of vertices $V \subseteq V_\Sigma$ for a system-level path $\pi := \{\pi_1, \ldots, \pi_n\} \in \mathbf{\Pi}^v_\Sigma$ and a duration Δ is

$$wl^+_{V,\pi}(\Delta) := \sum_{x=1}^{n} wl^+_{V,\pi_x}(\Delta) \tag{3}$$

with $$wl^+_{V,\pi_x}(\Delta) := cost(|pre_{\Delta+\mathbf{J}(fst(\pi_x))}(\pi_x)|_V)$$

We will show (see Lemma 2 in Sec. 6) that the workload of a set of vertices $V \subseteq V_\Sigma$ for any prefix of length Δ of a level-p busy window represented by a system path π is upper bounded by $wl^+_{V,\pi}(\Delta)$.

Step 1: Computing $\mathbf{\Pi}^v_\Sigma$ — *see Theorem 1 in Sec. 6*

Let $\Pi^v_x(\Delta)$ denote the set of paths π_x of task $G_x \in \Sigma$ which

- are the longest paths fitting in $\Delta + \mathbf{J}^+_x$, that is, such that $len(\pi_x) \leq \Delta + \mathbf{J}^+_x$ and for all valid suffixes s, $len(\pi_x \cdot s) > \Delta + \mathbf{J}^+_x$; where $\mathbf{J}^+_x$ representing the largest release jitter among V_x.
- if the vertex under study v belongs to G_i then we consider only paths π_x with occurrences of v.

With this notion of path, we compute a sufficiently large duration which can bound the length of any level-p busy window. That duration is the least positive fixed point, written $\mathbf{W}_k$, of the following equation

$$\Delta = N + \sum_{x=1}^{n} \max_{\pi_x \in \Pi^v_x(\Delta)} \{wl^+_{hep(p),\pi_x}(\Delta)\} \tag{4}$$

with N denoting the greatest non-preemptable segment in the system. At each iteration, the greatest workload among all possible paths within Δ is selected. The obtained fixed point is clearly an upper bound on the duration of any possible busy window. Therefore, to bound $wcrt(v)$, it is sufficient to examine all the following

combinations,

$$\mathbf{\Pi}_{\Sigma}^{v} := \bigtimes_{x=1}^{n} \Pi_{x}^{v}(\mathbf{W}_k) \tag{5}$$

where $\bigtimes_{x=1}^{n}$ denotes the Cartesian product of all paths of length bounded by $\mathbf{W}_k$ for the n tasks. Thus, any level-p busy window can be represented by (possibly a prefix of) a path in $\mathbf{\Pi}_{\Sigma}^{v}$.

Step 2: Computing upper bounds for each path in $\mathbf{\Pi}_{\Sigma}^{v}$

We now show how to compute $BW_{p,\pi}^{+}$, $q_{v,\pi_i,BW_{p,\pi}^{+}}^{+}$, $Q_{v,\pi}^{+}(q)$ and $\theta_{v,\pi_i}^{-}(q)$ for a given system path $\pi := \{\pi_1, \ldots, \pi_n\} \in \mathbf{\Pi}_{\Sigma}^{v}$.

a) *Computing $BW_{p,\pi}^{+}$ — see Theorem 2 in Sec. 6*

Having non-preemptable segments implies that vertices in $lp(p)$ may execute within a level-p busy window. Still, the definition of a level-p busy window implies that:

- at most one non-preemptable segment of a vertex in $lp(p)$ can execute in a level-p busy window; and
- such a segment (if it exists) must have started its execution before the beginning of the level-p busy window.

As a result, the maximum duration that vertices in $lp(p)$ can execute in a level-p busy window is upper bounded by:

$$B_p := \max_{\substack{v_x \in lp(p) \\ C \in \vec{\mathbf{C}}(v_x)}} (C - 1) \tag{6}$$

with the convention that $B_p := 0$ if $lp(p) = \emptyset$. Now, let $BW_{p,\pi}^{+}$ be the least positive fixed point of the following equation:

$$\Delta = B_p + wl_{hep(p),\pi}^{+}(\Delta) \tag{7}$$

Then $BW_{p,\pi}^{+}$ is an upper bound on the length of any possible level-p busy window represented by π.

Note that it may be pessimistic to use B_p to bound the workload from $lp(p)$ because the largest non-preemptable segment among $lp(p)$ may not contribute to any level-p busy window represented by π. On the other hand, it reduces the complexity of the analysis and simplifies its correctness proof[6].

b) *Computing $q_{v,\pi_i,BW_{p,\pi}^{+}}^{+}$*

Definition 12 implies that the number of jobs of type v in a busy window is equal to the number of v in its representing path. In addition, since $BW_{p,\pi}^{+}$ is an upper bound on the length of any possible level-p busy window represented by π, let $q_{v,\pi_i,BW_{p,\pi}^{+}}^{+}$ be the number of vertices v in the prefix $pre_{BW_{p,\pi}^{+}+\mathbf{J}(fst(\pi_i))}(\pi_i)$ of path π_i, then $q_{v,\pi_i,BW_{p,\pi}^{+}}^{+}$ is an upper bound on the number of jobs of type v released in any level-p busy window represented by π.

c) *Computing $Q_{v,\pi}^{+}(q)$ — see Theorem 3 in Sec. 6*

The q-th queueing prefix of jobs of type v in a level-p busy window represented by π spans the execution of:

- at most one non-preemptable segment from a vertex in $lp(p)$ which started before that busy window;
- all jobs of task G_i with the same priority as v released during the prefix, up to the q-th job of v in π_i (minus the cost of its last segment);
- the jobs of vertices from G_i in $hp(p)$ released during the prefix; and
- the jobs of vertices in $hep(p)$ released during the prefix, except those from G_i.

Let $Q_{v,\pi}^{+}(q)$ be the least positive fixed point of the following equation:

$$\begin{aligned} \Delta = B_p & \\ & + wl_{ep(p)\cap V_i,\pi}^{+}(len_{v}^{q}(\pi_i) + 1) - last(\vec{\mathbf{C}}(v)) + 1 \\ & + wl_{hp(p)\cap V_i,\pi}^{+}(\Delta) \\ & + wl_{hep(p)\setminus V_i,\pi}^{+}(\Delta) \end{aligned} \tag{8}$$

where $last(\vec{\mathbf{C}}(v))$ is the maximum cost of the last segment of v. Then, $Q_{v,\pi}^{+}(q)$ is an upper bound on the length of the q-th queueing prefix of jobs of type v in any possible level-p busy window represented by π.

d) *Computing $\theta_{v,\pi_i}^{-}(q)$ — see Theorem 4 in Sec. 6*

For each $q \leq q_{v,\pi_i,BW_{p,\pi}^{+}}^{+}$, a lower bound on the duration between t_1 and the q-th arrival of jobs of type v in any possible level-p busy window represented by π is:

$$\theta_{v,\pi_i}^{-}(q) := len_{v}^{q}(\pi_i) - \mathbf{J}(fst(\pi_i)) \tag{9}$$

e) *Computing $RT_{\pi}^{+}(v)$ — see Theorem 5 in Sec. 6*

The worst-case response time $RT_{\pi}^{+}(v)$ for path π can then be upper bounded based on the above upper and lower bounds.

$$RT_{\pi}^{+}(v) := \max_{q \leq q_{v,\pi_i,BW_{p,\pi}^{+}}^{+}} \{Q_{v,\pi}^{+}(q) - \theta_{v,\pi_i}^{-}(q) + last(\vec{\mathbf{C}}(v)) - 1\} \tag{10}$$

Step 3: Computing $RT_{\Sigma}^{+}(v)$

Finally, performing the same computation for all paths in $\mathbf{\Pi}_{\Sigma}^{v}$ yields the worst-case response time of jobs of type v.

$$RT_{\Sigma}^{+}(v) := \max_{\pi \in \mathbf{\Pi}_{\Sigma}^{v}} (RT_{\pi}^{+}(v)) \tag{11}$$

Note that the max functions in Equation 10 and 11 play different roles: The first one (Eq. 10) accounts for the fact that there may be several jobs of the same type (vertex) in a level-p busy window; the second one (Eq.11) allows a fine-grained analysis of dependencies (which are captured by the notion of path).

5.3 Improvements

For the sake of clarify, we have presented the analysis by first computing a superset of possible paths and then focusing on a single vertex. Clearly, this approach is not the most efficient. First, the paths considered are larger than needed and the analysis is likely to consider many time the same prefix common to many paths. Second, as presented, a complete system analysis would need to iterate the same process for each vertex. Both points entail costly and/or useless recomputations.

A more reasonable approach is to analyse pertinent paths only once. An analysis of the whole system would proceed by considering all possible alignments between vertices of all tasks. For each alignment $\{v_1, \ldots, v_n\}$, we compute the level-p busy window (with p the minimal priority) for all possible paths starting from the considered alignment. Paths are built on demand; they end when a

[6] An exact analysis would require considering up to $n + 1$ different critical instants for a system with n tasks.

level-p quiet time is reached. That busy window is the longest and includes all other busy windows. The computation should keep enough information to evaluate $Q^+_{v,\pi}(q)$, $\theta^-_{v,\pi_i}(q)$ and therefore $RT^+_\pi(v)$ for each vertex v in the path. Finally, the maximum of $RT^+_\pi(v)$ over all alignments gives the worst-case response time for any vertex v (*i.e.,* jobs of type v).

Further improvements can be made. Consider a vertex v_i in an alignment $\{v_1, \ldots, v_n\}$, then if there is some v_j with a lower priority that v_i whereas task G_j has another vertex with a priority equal or higher than v_i, then this alignment cannot be a worst case for v_i and $RT^+_\pi(v_i)$ does not need to be computed in that alignment.

The analysis described so far is precise; the only source of approximation lies in the blocking factor B_p (see Eq. 6). However, depending on the size of GD, considering all possible alignments may be overly expensive. Approximations should be studied. A possible approximation consists in computing a single over-approximated workload function for each priority and each task. This lower drastically the number of combinations to test (more on this in the Sec. 7). Note that this approach is similar to the approximation used by Tindell in his offset analysis [26] and by Guan *et al.* in their DRT analysis [15]. In fact, since our approach generalizes these two analyses, existing techniques should be applicable.

6 PROOF OF CORRECTNESS

We outline here the proofs of the main lemmas used to establish the correctness of the RTA presented in Sec. 5. Our short term goal is to complete these proofs using Coq and the Prosa library (see Section 7). In contrast with classical proofs in the RT community, machine-verified proofs require to list all used hypotheses, to specify formally concrete executions and to prove many properties usually taken for granted.

In the following, we consider a JFPLP schedule σ and a system-level job arrival sequence $\rho \sim \Sigma$ having a job $\jmath$ of type v with priority p and belonging to task G_i. Any such job occurs within a level-p busy window. Therefore, we consider that $\jmath$ occurs as the q-th job of type v in a level-p busy window starting at instant t_1.

6.1 Concrete busy window and queueing prefix

The key to the proof of the RTA is to show the correctness of the upper/lower bounds (*e.g.,* $BW^+_{p,\pi}$) computed using the abstract model (*i.e.,* GD). So, we need to specify formally the length BW_p of the considered level-p busy window and the length $Q_{v,\rho}(q)$ of the q-th queueing prefix of jobs of type v in order to bound them.

Length of the level-p busy window

The length of a level-p busy window starting at t_1, denoted by BW_p, can be computed as the least positive fixed point of the following equation.

$$\Delta = serv_{lp(p),\sigma}(t_1, \Delta) + wl_{hep(p),\rho}(t_1, \Delta) \tag{12}$$

The first term represents the service provided to the possible non preemptable segment of a lower priority job. Then, the amount of services performed by the scheduler for $hep(p)$ jobs is equal to the workload requested from $hep(p)$ within the duration of the busy window.

Length of the q-th queueing prefix

Similarly, computing the length of the q-th queueing prefix of jobs of type v (*i.e.,* the queueing prefix of $\jmath$) in the level-p busy window $[t_1, t_1 + BW_p[$ amounts to evaluate:

- the service provided to the possible non preemptable segment of a lower priority job;
- the workload of jobs of task G_i with the priority p (minus the cost of $\jmath$'s last segment);
- the workload of jobs of vertices from G_i in $hp(p)$
- the workload of other jobs with priorities in $hep(p)$.

Thus, the length of the q-th queueing prefix of jobs of type v, denoted by $Q_{v,\rho}(q)$, can be defined as the least positive fixed point of the following equation.

$$\begin{aligned}\Delta = &serv_{lp(p),\sigma}(t_1, \Delta)\\ &+wl_{ep(p)\cap V_i,\rho}(t_1, (\mathbf{a}(\jmath) - t_1 + 1)) - last(\vec{\mathbf{c}}(\jmath)) + 1\\ &+wl_{hp(p)\cap V_i,\rho}(t_1, \Delta)\\ &+wl_{hep(p)\setminus V_i,\rho}(t_1, \Delta)\end{aligned} \tag{13}$$

6.2 Basic lemmas

To bound BW_p and $Q_{v,\rho}(q)$ we rely on the following lemma.

Lemma 1 ($\star$). *Let $f, g : \mathbb{N} \to \mathbb{N}$ be two functions and Δ_1 and Δ_2 be fixed points of the equations $\Delta = f(\Delta)$ and $\Delta = g(\Delta)$ then, if for all $x : \mathbb{N}$, $f(x) \leq g(x)$ and, for all $x : \mathbb{N}^+$, $x < \Delta_1$, we have $x < f(x)$, then $\Delta_1 \leq \Delta_2$.*

PROOF. Easy proof by contradiction. □

Using this lemma the proof that $BW^+_{p,\pi}$ and $Q^+_{v,\pi}(q)$ upper bound BW_p and $Q_{v,\rho}(q)$ respectively, amounts to show that $serv$ and wl are increasing and to compare the *rhs* of their recursive definitions.

A key step to this aim is to bound the workload of a set of jobs of types in V during a busy window by the workload of the set of vertices in V of the path representing that busy window. More generally, we prove the following lemma:

Lemma 2. *Let Δ be a time interval and let $\pi := \{\pi_1, \ldots, \pi_n\}$ be the system path representing $\hat{\rho}_{/[t_1,t_1+\Delta[}$* i.e., *$\hat{\rho}_{/[t_1,t_1+\Delta[} \sim \pi$, then*

$$wl_{V,\rho}(t_1, \Delta) \leq wl^+_{V,\pi}(\Delta) \tag{14}$$

Furthermore, that bound is tight.

PROOF. The arrival sequence ρ can be decomposed into n independent task-level job arrival sequences $\{\rho_1, \ldots, \rho_n\}$ where each ρ_x is the arrival sequence of the jobs of task G_x. Then,

$$wl_{V,\rho}(t_1, \Delta) = \sum_{x=1}^{n} wl_{V,\rho_x}(t_1, \Delta)$$

Recall the definition of

$$wl^+_{V,\pi}(\Delta) := \sum_{x=1}^{n} cost(|pre_{\Delta+\mathbf{J}(fst(\pi_x))}(\pi_x)|_V)$$

Therefore, it is sufficient to prove, for all task G_x that

$$wl_{V,\rho_x}(t_1, \Delta) \leq cost(|pre_{\Delta+\mathbf{J}(fst(\pi_x))}(\pi_x)|_V)$$

We show that the workload of $\hat{\rho}_{x/[t_1,t_1+\Delta[}$ is maximal when:

(1) any two consecutive jobs of ρ_x are separated by their minimum inter-arrival time d;

(2) all jobs take their maximum cost (*i.e.,* WCET) $\mathbf{C}$;
(3) the first job in the interval releases at t_1 after having experienced its maximum release jitter ($\mathbf{J}(fst(\pi_x))$) whereas the jitter of all other jobs is null.

When these three conditions are met, the value of $wl_{V,\rho_x}(t_1, \Delta)$ is maximal and exactly equal to $cost(|pre_{\Delta+\mathbf{J}(fst(\pi_x))}(\pi_x)|_V)$. That directly implies Lemma 2. □

6.3 Correctness of $\mathbf{\Pi}^{\upsilon}_{\Sigma}$

As a first step towards the correctness proof of the RTA, we must show that any possible level-p busy window is represented by one path in $\mathbf{\Pi}^{\upsilon}_{\Sigma}$.

Theorem 1. *Any level-p busy window is represented by at least one path in $\mathbf{\Pi}^{\upsilon}_{\Sigma}$ as defined in Equation 5.*

Proof. It suffices to show that $\mathbf{W}_k$ bounds the length of any level-p busy window. *i.e.,*

$$BW_p \leq \mathbf{W}_k \tag{15}$$

Recall that $\mathbf{W}_k$ is the least positive fixed point of the equation

$$\Delta = N + \sum_{x=1}^{n} \max_{\pi_x \in \Pi^{\upsilon}_x(\Delta)} \{wl^+_{hep(p),\pi_x}(\Delta)\} \tag{16}$$

We first prove that the *rhs* of Equation 16 bounds the one of Equation 17. It is fairly easy to prove that $B_p \leq N$ since N represents the largest segment in the system. Furthermore, for each task $G_x \in \Sigma$, we clearly have

$$wl^+_{hep(p),\pi_x}(\Delta) \leq \max_{\pi_x \in \Pi^{\upsilon}_x(\Delta)} \{wl^+_{hep(p),\pi_x}(\Delta)\}$$

and therefore, for the system path and all tasks,

$$wl^+_{hep(p),\pi}(\Delta) \leq \sum_{x=1}^{n} \max_{\pi_x \in \Pi^{\upsilon}_x(\Delta)} \{wl^+_{hep(p),\pi_x}(\Delta)\}$$

Then, Lemma 1 entails $BW^+_{p,\pi} \leq \mathbf{W}_k$ and Theorem 2 permits to establish Equation 5 by transitivity. □

6.4 Correctness of bounds for job $\jmath$

Now, we show the correctness of the upper/lower bounds $BW^+_{p,\pi}$, $q^+_{\upsilon,\pi_i,BW^+_{p,\pi}}$, $Q^+_{\upsilon,\pi}(q)$ and $\theta^-_{\upsilon,\pi_i}(q)$ which implies the correctness of the upper-bound $RT^+_{\pi}(\upsilon)$ for the response time of $\jmath$. Let π be the path representing the level-p busy window $[t_1, t_1 + BW_p[$,

Theorem 2. *Let $BW^+_{p,\pi}$ be the least positive fixed point of the equation*

$$\Delta = B_p + wl^+_{hep(p),\pi}(\Delta) \tag{17}$$

then

$$BW_p \leq BW^+_{p,\pi} \tag{18}$$

Proof. We first prove that the *rhs* of Equation 17 bounds the *rhs* of

$$\Delta = serv_{lp(p),\sigma}(t_1, BW_p) + wl_{hep(p),\rho}(t_1, \Delta) \tag{19}$$

The definition of a level-p busy window implies that at most one non-preemptable segment of any vertex in $lp(p)$ of π can execute in $[t_1, t_1 + BW_p[$. Further, such a segment (if it exists) must have started execution before t_1. Therefore, $serv_{lp(p),\sigma}(t_1, BW_p)$ is bounded by B_p. Lemma 2 ensures that the second term of the *rhs* of equation 19 is bounded by $wl^+_{hep(p),\pi}(\Delta)$. Then, Lemma 1 permits to conclude. □

Lemma 3. *The number of vertices υ in π_i upper-bounds q:*

$$q \leq q^+_{\upsilon,\pi_i,BW^+_{p,\pi}} \tag{20}$$

Proof. This result follows by the definition of a path representing a busy window and $q^+_{\upsilon,\pi_i,BW^+_{p,\pi}}$. □

Theorem 3. *Let $Q^+_{\upsilon,\pi}(q)$ be the least positive fixed point of the equation*

$$\begin{aligned}\Delta = B_p \\ &+ wl^+_{ep(p)\cap V_i,\pi}(len^q_{\upsilon}(\pi_i) + 1) - last(\vec{\mathbf{C}}(\upsilon)) + 1 \\ &+ wl^+_{hp(p)\cap V_i,\pi}(\Delta) \\ &+ wl^+_{hep(p)\setminus V_i,\pi}(\Delta)\end{aligned} \tag{21}$$

then it bounds the length of the q-th queueing prefix of jobs of type υ in the level-p busy window i.e.,

$$Q_{\upsilon,\rho}(q) \leq Q^+_{\upsilon,\pi}(q) \tag{22}$$

Proof. As for Theorem 2, it suffices to prove that the *rhs* of Equation 21 bounds the *rhs* of Equation 13. We prove it by considering each term in turn.

(1) In Theorem 2, we proved $serv_{lp(p),\sigma}(t_1, BW_p) \leq B_p$. Further, the definition of a queueing prefix implies that $Q_{\upsilon,\rho}(q) \leq BW_p$. Therefore, for all $\Delta \leq Q_{\upsilon,\rho}(q)$ (which is sufficient for Lemma 1), $serv_{lp(p),\sigma}(t_1, \Delta) \leq serv_{lp(p),\sigma}(t_1, BW_p) \leq B_p$.
(2) The second term can be divided in two parts:

$$wl^+_{ep(p)\cap V_i,\pi\setminus\upsilon^q}(len^q_{\upsilon}(\pi_i) + 1) \tag{23}$$

$$+ \mathbf{C}(\upsilon) - last(\vec{\mathbf{C}}(\upsilon)) + 1 \tag{24}$$

where υ^q denotes the q-th υ in π.
The second term of equation 13 can be separated in two parts as well:

$$wl_{ep(p)\cap V_i,\rho\setminus\jmath}(t_1, \mathbf{a}(\jmath) - t_1 + 1) \tag{25}$$

$$+ \mathbf{c}(\jmath) - last(\vec{\mathbf{c}}(\jmath)) + 1 \tag{26}$$

Now, it is sufficient to prove that (25) ≤ (23) and (26) ≤ (24).

- According to Definition 12, the type of any job of $hep(p) \cap V_i$ released in $[t_1, \mathbf{a}(\jmath) + 1[$ occurs $pre_{\mathbf{J}(fst(\pi_i))+len^q_{\upsilon}(\pi_i)+1}(\pi_i)$. Taking into account the minimum inter-arrival time, we have

$$wl_{ep(p)\cap V_i,\rho}(t_1, \mathbf{a}(\jmath) - t_1 + 1) \leq wl^+_{ep(p)\cap V_i,\pi}(len^q_{\upsilon}(\pi_i) + 1)$$

  If we filter out $\jmath$ from ρ and the q-th υ (job $\jmath$'s type) from π, (25) ≤ (23) follows.
- By definition, we know that $\vec{\mathbf{c}}(\jmath) = \langle c_1, \ldots, c_s\rangle$, $\vec{\mathbf{C}}(\upsilon) = \langle C_1, \ldots, C_s\rangle$ and, for all $i = 1, \ldots, s$, $c_i \leq C_i$. So

$$\sum_{i=1}^{s-1} c_i \leq \sum_{i=1}^{s-1} C_i$$

  Equivalently $(\mathbf{c}(\jmath) - last(\vec{\mathbf{c}}(\jmath)) \leq (\mathbf{C}(\upsilon) - last(\vec{\mathbf{C}}(\upsilon))$. Therefore (26) ≤ (24) follows.

The last two inequalities between the last two terms of (21) and of (13) follow directly from Lemma 2. □

Theorem 4. *The term $\theta^{-}_{v,\pi_i}(q) = len^{q}_{v}(\pi_i) - \mathbf{J}(fst(\pi_i))$ is a lower bound of the duration between the arrival of $\jmath$ and the beginning of its level-p busy window.*

$$\theta^{-}_{v,\pi_i}(q) \leq \mathbf{a}(\jmath) - t_1 \tag{27}$$

Proof. By cases.
(1) $\mathbf{a}(\jmath) - \mathbf{t_1} < \mathbf{0}$. The job $\jmath$ arrives before t_1 but releases at or after t_1 *i.e.,* $t_1 \leq \mathbf{a}(\jmath) + \mathbf{j}(\jmath)$. Constrained jitter implies that no other job of the same task arrives before the release of $\jmath$. So job $\jmath$ must be the first vertex of π_i and $len^{q}_{v}(\pi_i) = 0$. By convention, $\mathbf{j}(\jmath) \leq \mathbf{J}(v)$, therefore $-\mathbf{J}(v) \leq -\mathbf{j}(\jmath) \leq \mathbf{a}(\jmath) - t_1$ and equation 27 holds.

(2) $\mathbf{a}(\jmath) - \mathbf{t_1} \geq \mathbf{0}$. Let $\jmath'$ be the job corresponding to the first vertex in π_i. We know $\mathbf{a}(\jmath) - \mathbf{a}(\jmath') \geq len^{q}_{v}(\pi_i)$ and $\mathbf{j}(\jmath') \leq \mathbf{J}(fst(\pi_i))$. Then, $len^{q}_{v}(\pi_i) - \mathbf{J}(fst(\pi_i)) \leq \mathbf{a}(\jmath) - \mathbf{a}(\jmath') - \mathbf{j}(\jmath')$ and since $\mathbf{a}(\jmath') + \mathbf{j}(\jmath') \geq t_1$ that is $-\mathbf{a}(\jmath') - \mathbf{j}(\jmath') \leq t_1$, equation 27 follows. □

Lemma 4. *Let $RT_\jmath$ be the response time of job $\jmath$ then*

$$RT_\jmath \leq Q^{+}_{v,\pi}(q) - \theta^{-}_{v,\pi_i}(q) + last(\vec{\mathbf{C}}(v)) - 1 \tag{28}$$

Proof. By definition $RT_\jmath := \mathbf{end}(\jmath) - \mathbf{a}(\jmath)$. Also, when a job begins to execute its last non-preemtable segment it cannot be preempted until its completion. Using the notion of q-th queueing prefix, we have

$$\mathbf{end}(\jmath) = t_1 + Q_{v,\rho}(q) + last(\vec{\mathbf{c}}(\jmath)) - 1$$

Note that $Q_{v,\rho}(q)$ includes the first cost unit of job $\jmath$'s last segment, so we should subtract 1 from $last(\vec{\mathbf{c}}(\jmath))$. Equivalently, we have

$$RT_\jmath = Q_{v,\rho}(q) - (\mathbf{a}(\jmath) - t_1) + last(\vec{\mathbf{c}}(\jmath)) - 1$$

The result follows from Theorem 3, Theorem 4 and the fact that costs in the model upper-bounds costs in the execution. □

6.5 Correctness of the RTA

The previous result applies to an arbitrary job $\jmath$ of type v in a busy window starting at t_1. It can be used to upper-bound the response times of any job of type v in that busy window, and by extension any job of type v in the arrival sequence.

Theorem 5. *The response time of any job of type v released in a busy window starting at t_1 is bounded by*

$$\max_{q \leq q^{+}_{v,\pi_i,BW^{+}_{p,\pi}}} \{Q^{+}_{v,\pi}(q) - \theta^{-}_{v,\pi_i}(q) + last(\vec{\mathbf{C}}(v)) - 1\} \tag{29}$$

Proof. Follows from properties of max and Lemma 4. □

Theorem 6. *The response time of any job of type v released in any arrival sequence $\rho \sim \Sigma$ is bounded by*

$$\max_{\pi \in \mathbf{\Pi}^{v}_{\Sigma}} \left\{ \max_{q \leq q^{+}_{v,\pi_i,BW^{+}_{p,\pi}}} \left\{ Q^{+}_{v,\pi}(q) - \theta^{-}_{v,\pi_i}(q) + last(\vec{\mathbf{C}}(v)) - 1 \right\} \right\} \tag{30}$$

Proof. Follows from properties of max and Theorem 5. □

7 DISCUSSION

In this section, we discuss the significance of the presented model and analysis in the context of our broader effort toward a Coq library of schedulability results.

In the past decades, there have a few attempts at proving methods [13, 14] for solving real-time problems. Recently, the Prosa library [11, 12] has been proposed to provide formal specifications and mechanized proofs for schedulability analyses using the Coq proof assistant. The motivation behind our general task model for fixed priority scheduling is to add it to the Prosa library and prove the correctness of its RTA. It can thus cover a large variety of existing models and analyses.

7.1 Proving in Coq the RTA of GD systems

The complete Coq proof of the RTA for GD systems is still in progress[7] and can be separated into two parts:

(1) The generic proof of RTAs for the JFPLP scheduling policy. For this part, many definitions (*i.e.,* those in Sec. 2) have been formalized and used in Prosa, as well as a significant part of the proof. We are actually formalizing the proof of a more general statement, which does not rely on a task model, but on an abstract workload function

$$wl^{+} : (\rho \to \mathbb{N} \to \mathbb{N} \to T) \to \mathbb{N} \to \mathbb{N}$$

where: (a) the first argument $(\rho \to \mathbb{N} \to \mathbb{N} \to T)$ denotes a function taking a job arrival sequence, a time instant and a time duration and returning an abstract *candidate* represented by the type T, where candidates correspond to incomparable scenarios which must be analyzed, e.g. paths for the GD model; (b) the second argument denotes a time duration such that wl^{+} returns the workload during that duration. The proof as well as the analysis are then applicable to many task models respecting the fixed priority scheduling policy, including the GD model, by instantiating that function.

(2) Specifying the GD task model and instantiating the function wl^{+}. This part is not formalized yet; however, according to our experience, it should not raise any issue.

7.2 Intended use of the analysis

One of our objectives is to formally certify our RTA in order to:

- compare it with existing RTAs for task models which can be expressed by GD, in terms of precision and time complexity;
- obtain novel machine-verified RTAs which take into account *e.g.,* jitter, non-preemptable segments and offsets;
- reuse the generic part of the proof to propose machine-verified RTAs for task models beyond GD by focusing on upper bounding the workload.

7.3 Beyond the current analysis

By proposing a unified analysis for models as different as the DRT model and Tindell's offset model, our work underlines the generic parts of the proof structure of such RTAs. Based on this, we can now propose a framework which formalizes these steps in a generic manner, to be reused for any new task model. Such steps include

[7] For more information, please visit https://team.inria.fr/spades/generalized-digraph/.

the use of sustainability properties [5, 11], but also strategies to efficiently approximate the worst-case response time.

8 RELATED WORK

Many task models have been proposed to analyze different fixed-priority scheduling policies. Depending on their capacity to model intra- or inter-task dependencies, those models can be divided into three categories.

The simplest models do not consider any kind of dependency: there is only one type of job for each task. The classic *periodic* task model, presented by Liu and Layland [16] characterizes a task by its worst-case execution time C and its activation period P. The *sporadic* task model [19] generalizes activation periods of tasks by introducing the concept of minimum inter-arrival time. Later Thiele et *al.* introduced the *real-time calculus* [24], whose *arrival curves* can model many more arrival patterns. Another category of models considers *intra-task* dependencies: there may be several types of jobs for each task. The *multiframe* model [18], characterizes a task by an array of execution times $(C^0, C^1, \ldots, C^{N-1})$ and a minimum inter-arrival time P. A task has N types of jobs and the $(i+1)$-th job in the arrival sequence has the worst-case execution time $C^{(i \mod N)}$ and arrives at least P time units after the arrival time of the i-th job. The model $(\boldsymbol{C}, \boldsymbol{P}, \boldsymbol{D})$ extends the multiframe model by allowing each type of job to have a different inter-arrival time and deadline [6]. Later, Baruah introduced the *recurring branching* task model [7], which adds branching structures. Each task can be represented by a tree of job types $(\boldsymbol{C}, \boldsymbol{D})$ labeled by minimum inter-arrival times. Two other extensions use *directed acyclic graphs* instead of trees: the *recurring real-time* task model [8, 9]), and the *non-cyclic recurring real-time* task model [4]. More recently, Stigge et *al.* introduced the DRT task model [21] and its extended version [22] that use arbitrary graphs. None of those models allows to model inter-task dependencies.

One of the most classic task model taking *inter-task* dependencies into account is Tindell's *offsets* model. A system is made of a collection of transactions regrouping periodic tasks having fixed timing relations. More recently, the DRT model was extended by Mohaqeqi et al. to allow the RDV mechanism [17] and by Abdullah et al. to take into account shared resources [3].

To the best of our knowledge, none of the previous models is general enough to express at the same time intra- and inter-task dependencies as well as arrival curves. Our model is a generalization in this respect. The formal certification of its associated RTA should permit to factorize the correctness proofs of many analyses.

9 CONCLUSION

In this paper, we have introduced the GD model, a generalization of the DRT task model that is expressive enough to model and analyze many different fixed-priority systems. In particular, GD can express dependencies between jobs as well as tasks. The work presented in this paper is motivated by our ongoing contribution to Prosa, a Coq library of models and analyses of real-time systems. The GD model and its associated RTA provide the needed foundations for a Coq response time analysis of complex systems, in particular regarding dependencies. Future work includes:

- The complete Coq formalization of the presented RTA, which is still in progress.
- A formal comparison of our proposed analysis with the existing RTAs of specific types of DRTs, *e.g.,* constrained deadline under job-level FPP or task-level FPNP [23], and arbitrary deadline under task-level FPP [20]. Indeed, our RTA for GD uses a queueing prefix technique which may require a smaller number of paths to be analyzed.
- A practical study of the complexity of the analysis and of the possible trade-off between accuracy of the computed bounds and runtime performance of an RTA implementation.
- Extensions to more complex models, in particular to task chains and multiprocessor systems.
- A theoretical connection between the RTA proposed here and the notion of sustainability.

We believe that this work represents a significant step toward integrating previously independent features into a unified framework for the response time analysis of real-time systems.

ACKNOWLEDGMENT

This work has been partially supported by the LabEx PERSYVAL-Lab (grant ANR-11-LABX-0025-01) through the CASERM project and the French national research organization ANR (grants ANR-15-CE25-0008 and ANR-17-CE25-0016) through the VOCAL and RT-PROOFS projects. We would like to thank Maxime Lesourd for insightful discussions.

REFERENCES

[1] A Library for formally proven schedulability analysis. http://prosa.mpi-sws.org/.
[2] The Coq proof assistant. http://coq.inria.fr.
[3] J. Abdullah, M. Mohaqeqi, G. Dai, and W. Yi. Schedulability analysis and software synthesis for graph-based task models with resource sharing. RTAS, 2018.
[4] S. Baruah. The non-cyclic recurring real-time task model. In *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*, pages 173–182. IEEE, 2010.
[5] S. Baruah and A. Burns. Sustainable scheduling analysis. In *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 159–168, Dec 2006.
[6] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized multiframe tasks. *Real-Time Systems*, 17(1):5–22, 1999.
[7] S. K. Baruah. Feasibility analysis of recurring branching tasks. In *Real-Time Systems, 1998. Proceedings. 10th Euromicro Workshop on*, pages 138–145. IEEE, 1998.
[8] S. K. Baruah. A general model for recurring real-time tasks. In *Real-Time Systems Symposium, 1998. Proceedings. The 19th IEEE*, pages 114–122. IEEE, 1998.
[9] S. K. Baruah. Dynamic-and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24(1):93–128, 2003.
[10] A. Bouillard and É. Thierry. An algorithmic toolbox for network calculus. *Discrete Event Dynamic Systems*, 18(1):3–49, Mar 2008.
[11] F. Cerqueira, G. Nelissen, and B. B. Brandenburg. On Strong and Weak Sustainability, with an Application to Self-Suspending Real-Time Tasks. In *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, volume 106 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:21, 2018.
[12] F. Cerqueira, F. Stutz, and B. B. Brandenburg. Prosa: A case for readable mechanized schedulability analysis. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 273–284, July 2016.
[13] M. Cordovilla, F. Boniol, E. Noulard, and C. Pagetti. Multiprocessor schedulability analyser. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 735–741. ACM, 2011.
[14] E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Schedulability analysis of fixed-priority systems using timed automata. *Theoretical Computer Science*, 354(2):301–317, 2006.
[15] N. Guan, C. Gu, M. Stigge, Q. Deng, and W. Yi. Approximate response time analysis of real-time task graphs. In *2014 IEEE Real-Time Systems Symposium*, pages 304–313, Dec 2014.
[16] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.
[17] M. Mohaqeqi, J. Abdullah, N. Guan, and W. Yi. Schedulability analysis of synchronous digraph real-time tasks. In *Real-Time Systems (ECRTS), 2016 28th Euromicro*

Conference on, pages 176–186. IEEE, 2016.

[18] A. K. Mok and D. Chen. A multiframe model for real-time tasks. In *17th IEEE Real-Time Systems Symposium*, pages 22–29, Dec 1996.

[19] A. K.-L. Mok. *Fundamental design problems of distributed systems for the hard-real-time environment.* PhD thesis, Massachusetts Institute of Technology, 1983.

[20] C. Peng and H. Zeng. Response time analysis of digraph real-time tasks scheduled with static priority: generalization, approximation, and improvement. *Real-Time Systems*, 54(1):91–131, Jan 2018.

[21] M. Stigge, P. Ekberg, N. Guan, and W. Yi. The digraph real-time task model. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, pages 71–80. IEEE, 2011.

[22] M. Stigge, P. Ekberg, N. Guan, and W. Yi. On the tractability of digraph-based task models. In *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*, pages 162–171. IEEE, 2011.

[23] M. Stigge and W. Yi. Combinatorial abstraction refinement for feasibility analysis of static priorities. *Real-Time Systems*, 51(6):639–674, 2015.

[24] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, volume 4, pages 101–104. IEEE, 2000.

[25] K. Tindell. *Using offset information to analyse static priority pre-emptively scheduled task sets.* Technical report YCS 182. University of York, Department of Computer Science, 1992.

[26] K. Tindell. *Adding time-offsets to schedulability analysis.* University of York, Department of Computer Science, 1994.